Data Exploration and PreProcessing

In [46]:
```
import pandas as pd
import numpy as np
```

In [3]:
```
df=pd.read_csv("heart.csv")
df
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	targe
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	(
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	(
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	(
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	(
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	'
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	(
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	(
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	'
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	(

1025 rows × 14 columns

In [4]:
```
df.head()
```

Out[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

In [5]:
```
#EDA
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

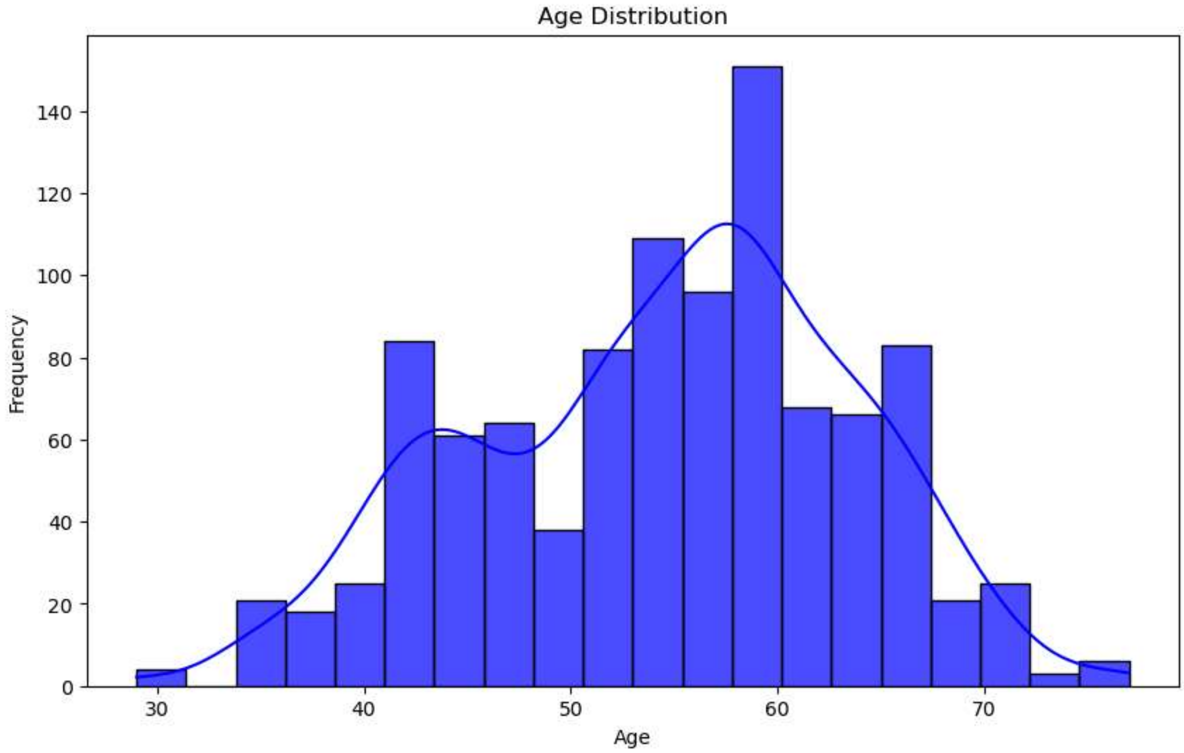
In [7]:
```
df.describe()
```

Out[7]:

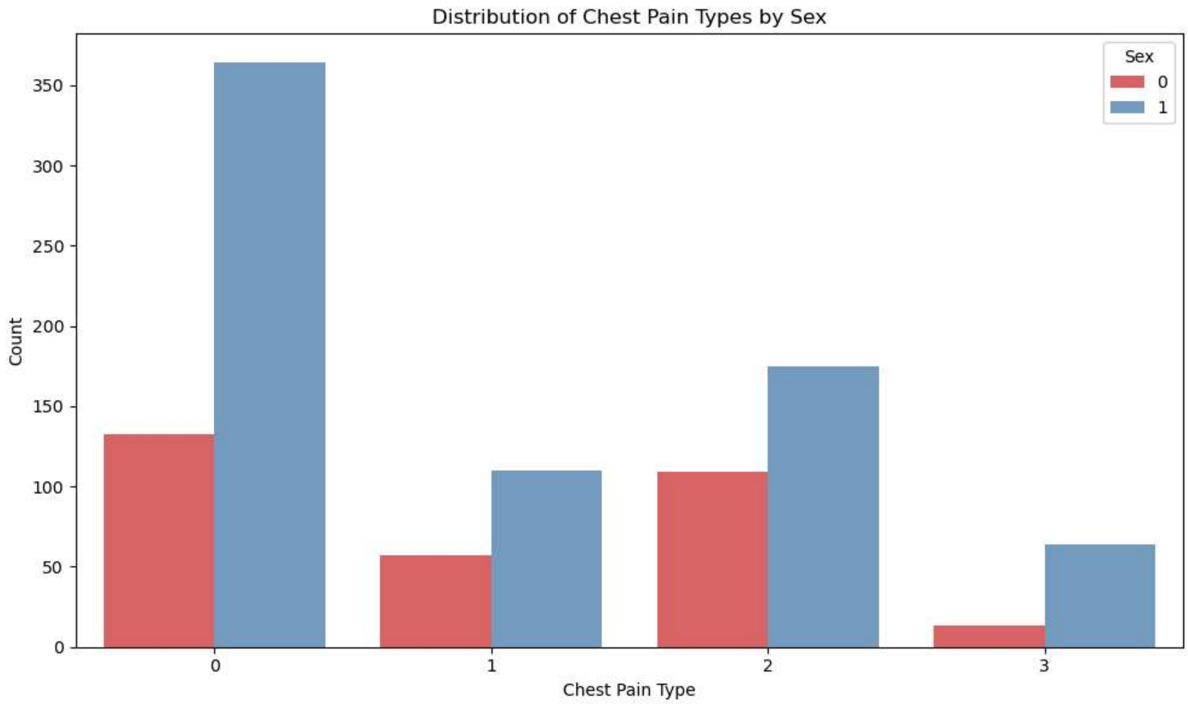
	age	sex	cp	trestbps	chol	fbs	restecg
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000

In [8]:
```
plt.figure(figsize=(10, 6))
sns.histplot(df['age'], bins=20, color='blue', kde=True, alpha=0.7)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
```

Out[8]:
```
Text(0, 0.5, 'Frequency')
```



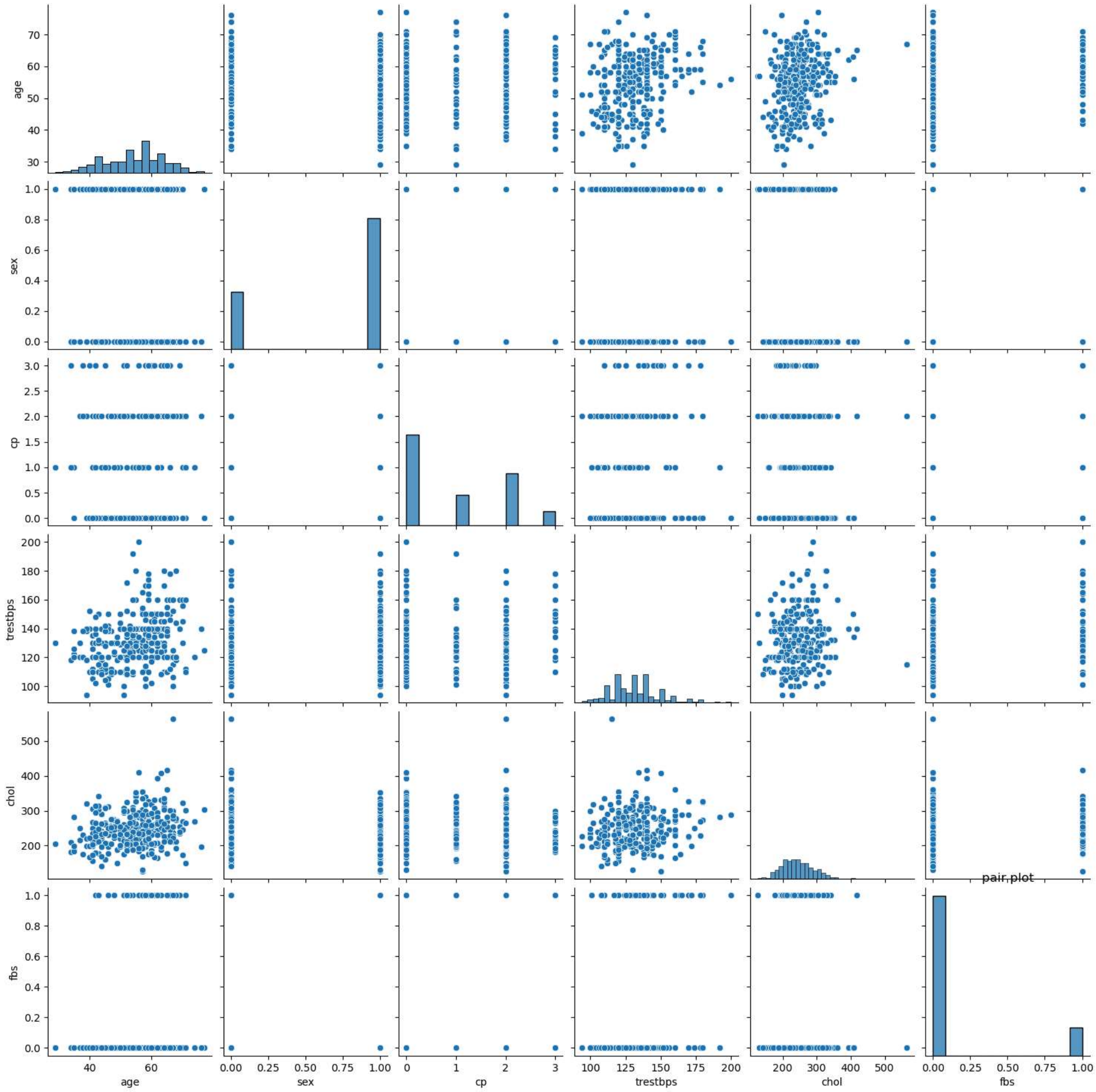
In [9]:
```
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='cp', hue='sex', palette='Set1', alpha=0.75)
plt.title('Distribution of Chest Pain Types by Sex')
plt.xlabel('Chest Pain Type')
plt.ylabel('Count')
plt.legend(title='Sex')
plt.tight_layout()
plt.show()
```



In [10]:
```
plt.figure(figsize=(15, 8))
sns.heatmap(df.corr(), annot=True)
```

Out[10]: <Axes: >

In [11]:
```python
subset=df[['age','sex','cp','trestbps','chol','fbs']]
sns.pairplot(subset)
plt.title('pair plot')
plt.show()
```



In [12]:
```
X = df.drop('target', axis=1)
y = df['target']
```

Feature Engineering

In [13]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, rar
```

In [15]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

Model Selection and Training

In [16]:
```
#LogisticRegression
LR = LogisticRegression()
LR.fit(X_train,y_train)
Y_pred_LR = LR.predict(X_test)
```

In [17]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusic
```

In [18]:
```
Accuracy =accuracy_score(y_test, Y_pred_LR)
Precision=precision_score(y_test, Y_pred_LR)
Recall = recall_score(y_test, Y_pred_LR)
Confusion_Matrix = confusion_matrix(y_test, Y_pred_LR)
```

In [19]:
```
Accuracy , Precision , Recall , Confusion_Matrix
```

Out[19]:
```
(0.7951219512195122,
 0.7563025210084033,
 0.8737864077669902,
 array([[73, 29],
        [13, 90]], dtype=int64))
```

In [20]:
```
#RandomForestClassifier
RF = RandomForestClassifier()
RF.fit(X_train,y_train)
Y_pred_RF = RF.predict(X_test)
```

In [21]:
```
Accuracy =accuracy_score(y_test, Y_pred_RF)
Precision=precision_score(y_test, Y_pred_RF)
Recall = recall_score(y_test, Y_pred_RF)
Confusion_Matrix = confusion_matrix(y_test, Y_pred_RF)
```

In [22]:
```
Accuracy , Precision , Recall , Confusion_Matrix
```

Out[22]:
```
(0.9853658536585366,
 1.0,
 0.970873786407767,
 array([[102,   0],
        [  3, 100]], dtype=int64))
```

In [33]:
```
#SVC
svc = SVC()
svc.fit(X_train,y_train)
Y_pred_svc = svc.predict(X_test)
```

In [34]:
```
Accuracy =accuracy_score(y_test, Y_pred_svc)
Precision=precision_score(y_test, Y_pred_svc)
Recall = recall_score(y_test, Y_pred_svc)
Confusion_Matrix = confusion_matrix(y_test, Y_pred_svc)
```

In [35]:
```
Accuracy , Precision , Recall , Confusion_Matrix
```

Out[35]:
```
(0.8878048780487805,
 0.8508771929824561,
 0.941747572815534,
 array([[85, 17],
        [ 6, 97]], dtype=int64))
```

Hyperparameter Tuning

In [36]:
```
from sklearn.model_selection import GridSearchCV
```

In [37]:
```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10],
}
grid_rf = GridSearchCV(RandomForestClassifier(), param_grid, cv=3)
grid_rf.fit(X_train, y_train)

best_model = grid_rf.best_estimator_
y_pred_best = best_model.predict(X_test)
```

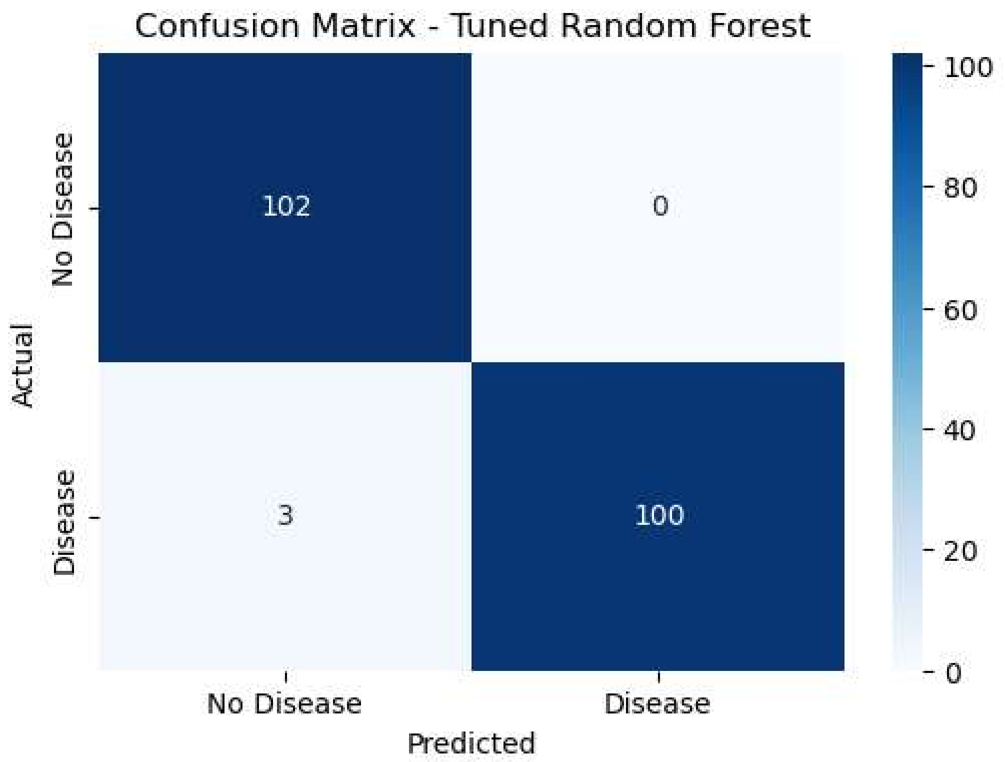
Model Selection and Explanation

In [38]:
```
Accuracy =accuracy_score(y_test, y_pred_best)
Precision=precision_score(y_test, y_pred_best)
Recall = recall_score(y_test, y_pred_best)
Confusion_Matrix = confusion_matrix(y_test, y_pred_best)
```

In [39]:
```
Accuracy , Precision , Recall , Confusion_Matrix
```

Out[39]:
```
(0.9853658536585366,
 1.0,
 0.970873786407767,
 array([[102,   0],
        [  3, 100]], dtype=int64))
```

In [40]:
```
plt.figure(figsize=(6, 4))
sns.heatmap(Confusion_Matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No D
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Tuned Random Forest')
plt.show()
```



In [41]:
```python
Classification_results = pd.DataFrame({
    'Models':['LogisticRegression','RandomForestClassifier','SVC'],
    'Accuracy': [ 0.7951219512195122 , 0.9853658536585366,0.8878048780487805],
    'Precision': [ 0.7563025210084033, 1.0,  0.8508771929824561],
    'Recall': [0.8737864077669902 , 0.970873786407767,  0.941747572815534],


})
Classification_results
```

Out[41]:

	Models	Accuracy	Precision	Recall
0	LogisticRegression	0.795122	0.756303	0.873786
1	RandomForestClassifier	0.985366	1.000000	0.970874
2	SVC	0.887805	0.850877	0.941748

In [50]:
```python
sample = X_test[0].reshape(1, -1)

sample_scaled = scaler.transform(sample)
prediction = RF.predict(sample_scaled)[0]

print("Predicted Heart Disease Risk:", "Yes" if prediction == 1 else "No")
```

```
Predicted Heart Disease Risk: Yes
C:\Users\damma\anaconda4\Lib\site-packages\sklearn\base.py:464: UserWarning: X doe
s not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

Brief Explanation

While working on this project, I learned that some health factors like chest pain type, age, and cholesterol levels play a big role in predicting heart disease. Visualizing the data helped me understand how these features are related. I also realized that using models like Random Forest gives better results because it can handle more complex patterns in the data.

Preprocessing steps like scaling and encoding were really important too—they made the models work more effectively.